

Fréchet Distance for Curves, Revisited*

Boris Aronov[†] Sariel Har-Peled[‡] Christian Knauer[§] Yusu Wang[¶]

Carola Wenk^{||}

April 30, 2015

Abstract

We revisit the problem of computing Fréchet distance between polygonal curves under L_1 , L_2 , and L_∞ norms, focusing on discrete Fréchet distance, where only distance between vertices is considered. We develop efficient algorithms for two natural classes of curves. In particular, given two polygonal curves of n vertices each, a ε -approximation of their discrete Fréchet distance can be computed in roughly $O(n\kappa^3 \log n/\varepsilon^3)$ time in three dimensions, if one of the curves is κ -bounded. Previously, only a κ -approximation algorithm was known. If both curves are the so-called *backbone curves*, which are widely used to model protein backbones in molecular biology, we can ε -approximate their Fréchet distance in near linear time in two dimensions, and in roughly $O(n^{4/3} \log nm)$ time in three dimensions. In the second part, we propose a pseudo-output-sensitive algorithm for computing Fréchet distance exactly. The complexity of the algorithm is a function of a quantity we call the *number of switching cells*, which is quadratic in the worst case, but tends to be much smaller in practice.

1 Introduction

Fréchet metric is a natural measure of similarity between two curves [EGH⁺02]. An intuitive definition of the Fréchet distance is to imagine that a dog and its handler are walking on their respective curves. Both can control their speed but can only go forward. The Fréchet distance of these two curves is the minimal length of any leash necessary for the handler and the dog to move from the starting points of the two curves to their respective endpoints. Fréchet distance and its variants have been widely used in many applications such as in dynamic time-warping [KP99], speech recognition [KHM⁺98], signature verification [PP90], and matching of time series in databases [KKS05].

*A preliminary version of this paper appeared in ESA 2006 [AHK⁺06].

[†]Dept. of Comp. Sci. & Engineering; Polytechnic School of Engineering; New York University, NY; Research supported in part by NSF ITR Grant CCR-00-81964 and by a grant from US-Israel Binational Science Foundation. <http://cis.poly.edu/~aronov>.

[‡]Dept. of Comp. Sci., University of Illinois; 1304 West Springfield Ave., Urbana, IL 61801; sariel@uiuc.edu. <http://sarielhp.org>.

[§]Universität Bayreuth; Institut für Angewandte Informatik; 95440 Bayreuth, Germany; christian.knauer@uni-bayreuth.de.

[¶]Dept. of Comp. Sci. and Engineering, The Ohio State Univ, Columbus, OH 43016; yusu@cse.ohio-state.edu. <http://www.cse.ohio-state.edu/~yusu/>.

^{||}Department of Computer Science, Tulane University, New Orleans, LA 70118, cwenk@tulane.edu, <http://www.cs.tulane.edu/~carola>.

Alt et al. [AG95] present an algorithm to compute the Fréchet distance between two polygonal curves of n and m vertices, respectively, in time $O(nm \log^2(nm))$. Improving this roughly quadratic-time solution for general curves seems to be hard, and so far, no algorithm, exact or approximate, with running time significantly smaller than $O(nm)$ has been found for this problem for general curves. Since the Fréchet distance essentially requires computing a correspondence between the two curves, it has some resemblance to the edit distance problem (which asks for the best alignment of two strings), for which no substantially subquadratic algorithm is known either.

On the other hand, another similarity measure, the *Hausdorff distance*, can be computed faster in the plane and approximated efficiently in higher dimensions. Unfortunately, Hausdorff distance does not reflect curve similarity well (see Figure 1 (a) for an example). Alt et al. [AKW04] showed that the Hausdorff distance and the Fréchet distance are the same for a pair of closed convex curves. They also showed that the two measures are closely related for κ -bounded curves. Roughly speaking, for any two points p, q on a κ -bounded curve τ , $\tau(p, q)$, the subcurve from p to q , is contained within some neighborhood of p and q with size roughly $\kappa \|p - q\|$ (see Figure 1 (b); precise definition is introduced later). Alt et al. showed that the Fréchet distance between any two κ -bounded curves is bounded by $\kappa + 1$ times the Hausdorff distance between them. This leads to a κ -approximation algorithm for the Fréchet distance for any pair of κ -bounded curves, and they also developed an algorithm to compute the reparametrizations of input curves that realize this approximation (see the definitions below). The algorithm runs in $O((n + m) \log^2(n + m) 2^{\alpha(n+m)})$ time in two dimensions. In three or higher dimensions, the time complexity is dominated by the computation of Hausdorff distance between the curves. Not much is known about the Fréchet distance for other types of curves. In fact, even for x -monotone curves in three dimensions, no known algorithm runs in substantially subquadratic time.

The problem of minimizing Fréchet distance under various classes of transformations has also been studied [AKW01, Wen02]. However, even in two dimensions, the exact algorithm takes roughly $O(n^6)$ time for computing the best Fréchet distance under translations, and roughly $O(n^8)$ time under rigid motions. Approximation algorithms have been studied [CM05, Wen02], but practical solutions remain elusive. The basic building block of those algorithms, as well as one of the bottlenecks, is the computation (or approximation) of the Fréchet distance between curves π and σ .

There is a slightly simpler version of the Fréchet distance, the *discrete Fréchet distance*, which only consider vertices of polygonal curves. Its computation takes $\Theta(n^2)$ time and space using dynamic programming [EM94], and no substantially subquadratic algorithm is known either. Fréchet distance has also been extended to graphs (maps) [AERW03], to piecewise smooth curves [Rot05], to simple polygons [BBW06] and to surfaces [AB05]. Finally, Fréchet distance was used as the similarity measure for morphing [EGHM01] between curves, and for high-dimensional approximate nearest neighbor search [Ind02]. It was also used for efficient curve simplification [AHMW05].

Our results. Given the apparent difficulty of improving the worst-case time complexity of computing the Fréchet distance between two unrestricted polygonal curves, we aim at developing algorithms for more realistic cases. First, in Section 3, we consider efficient approximation algorithms for the slightly simpler variant of Fréchet distance, the *discrete Fréchet distance*, the best algorithms for which currently have only slightly better worst-case time complexity than the continuous case. Most currently algorithms for computing Fréchet distance rely on a so-called *decision* procedure which determines whether a given distance is larger or smaller than the Fréchet distance between the two given curves. We observe that an approximation solution to the decision problem can lead to an approximation of Fréchet distance, and curve simplification can help us to approximate the decision problem efficiently. We apply this idea for two families of common curves. In the first case, given two polygonal curves of size n and m respectively,

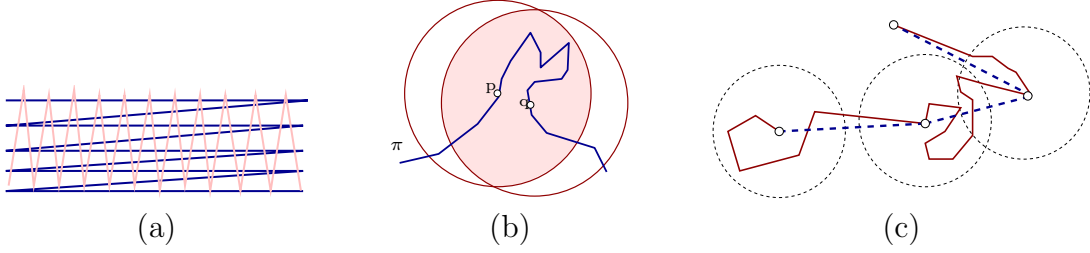


Figure 1: (a) Light and dark curves are close under Hausdorff but far under Fréchet distance. (b) π is κ -bounded iff for any $p, q \in \pi$, subchain $\pi(p, q)$ lies inside the shaded region, where the radius of two disks (centered at p and q) is $\kappa d(p, q)/2$. (c) The dashed curve μ -simplifies the solid one; the radius of each disk is μ .

with one of them being κ -bounded, we can ε -approximate their discrete Fréchet distance in $O((m + n\kappa^d/\varepsilon^d) \log(nm))$ time in d -dimensions. In the second case, both curves are so-called *backbone curves*, used widely to model molecular structures like protein backbones, and DNA/RNAs. We ε -approximate their (both *discrete* and *continuous*) Fréchet distance in near linear time in two dimensions, and in roughly $O(nm^{1/3} \log nm)$ time in three dimensions.

In Section 4, we shift our focus back to the exact computation of discrete Fréchet distance. Previously, the problem of deciding whether Fréchet distance was smaller than some threshold was cast as finding some viable path in the so-called *free-space diagram* which is a $n \times m$ map. We observe that such viable path can be computed once some subset \S of cells in the free-space diagram are given. The size of \S is nm in worst case, but is expected to be much smaller in general. Based on this observation, we present algorithms that run in $O(|\S| + n \log^{d-1} n)$ time for discrete Fréchet distance in L_∞ norm in d -dimensions. For L_2 norm, it takes roughly $O(|\S| + n^{4/3} \text{polylog } n)$ time in two dimensions, and $O(|\S| + n^{2-1/2^d} \text{polylog } n)$ time for $d > 2$.

2 Preliminaries

A (parameterized) curve in \mathbb{R}^d can be represented as a function $f: [0, 1] \rightarrow \mathbb{R}^d$. A (monotone) reparametrization α is a continuous non-decreasing function $\alpha: [0, 1] \rightarrow [0, 1]$ with $\alpha(0) = 0$ and $\alpha(1) = 1$. Given two curves $f, g: [0, 1] \rightarrow \mathbb{R}^d$, the *Fréchet distance* between them, $\delta_F(f, g)$, is defined as

$$\delta_F(f, g) := \inf_{\alpha, \beta} \max_{t \in [0, 1]} d(f(\alpha(t)), g(\beta(t))).$$

where $d(x, y)$ denotes the Euclidean distance between points x and y , and α and β range over all monotone reparametrizations.

Discrete Fréchet Distance. A simpler variant of the Fréchet distance for two polygonal curves $\pi = \langle p_1, p_2, \dots, p_n \rangle$ and $\sigma = \langle q_1, q_2, \dots, q_m \rangle$ is the *discrete Fréchet distance*, denoted by $\delta_D(\pi, \sigma)$. Imagine that both the dog and its handler can only stop at vertices of π and σ , and at any step, each of them can either stay at their current vertex or jump to the next one (i.e., magically, both the dog and the handler seem to have turned into frog princess and prince, respectively). The discrete Fréchet distance is defined as the minimal leash necessary at these discrete moments.

To formally define the discrete Fréchet distance, we first consider a discrete analog of continuous reparametrizations. A *discrete monotone reparametrization* α from $\{1, \dots, k\}$ to $\{1, \dots, \ell\}$ is a non-decreasing function $\alpha: \{1, \dots, k\} \rightarrow \{1, \dots, \ell\}$, for integers $k \geq \ell \geq 1$, with $\alpha(1) = 1, \alpha(k) = \ell$ and

$\alpha(i+1) \leq \alpha(i) + 1$, for all $i = 1, \dots, k-1$. An (*order-preserving complete*) *correspondence* between π and σ is a pair (α, β) of discrete monotone reparametrizations from $\{1, \dots, k\}$ to $\{1, \dots, m\}$ and $\{1, \dots, n\}$. The *discrete Fréchet distance* between π and σ , $\delta_D(\pi, \sigma)$ is

$$\delta_D(f, g) := \min_{(\alpha, \beta)} \max_{t \in [1, k]} d(f(\alpha(t)), g(\beta(t))),$$

where (α, β) range over all order-preserving complete correspondences between π and σ . An equivalent definition of order-preserving complete correspondence between π and σ is a set of pairs $M \subseteq \{(p, q) \mid p \in \pi, q \in \sigma\}$ such that (i) *order-preserving*: if $(p_i, q_j) \in M$, then no $(p_s, q_t) \in M$ for $s < i$ and $t > j$; and (ii) *complete*: for any $p \in \pi$ (resp. $q \in \sigma$), there exists some pair involving p (resp. q) in M . The discrete Fréchet distance is related to the edit distance between the “strings” π and σ where the cost of changing a symbol is the Euclidean distance of the relevant points.

It is well known that discrete and continuous versions of the Fréchet distance relate to each other as follows:

$$\delta_F(\pi, \sigma) \leq \delta_D(\pi, \sigma) \leq \delta_F(\pi, \sigma) + \max\{\ell_1, \ell_2\},$$

where ℓ_1 and ℓ_2 are the lengths of the longest edges in π and σ , respectively. This suggests using δ_D to approximate δ_F . Unfortunately, it seems that computing $\delta_D(\pi, \sigma)$ is asymptotically almost as hard as computing $\delta_F(\pi, \sigma)$.

Decision problem. In the original paper, Alt and Godau [AG95] used the following framework to compute $\delta_F(\pi, \sigma)$: First, develop a procedure that answers the following *decision problem* in $\Theta(nm)$ time and space by a dynamic programming algorithm: Given a parameter $\delta \geq 0$, is $\delta_F(\pi, \sigma) \leq \delta$? This procedure is then used as a subroutine to search for $\delta_F(\pi, \sigma)$ using parametric search paradigm within $O(nm \log^2 nm)$ time [AG95, AST94]. The same paradigm can be used to compute $\delta_D(\pi, \sigma)$ in $O(nm \log(nm))$ time by replacing the parametric search to a binary search. Although this is slightly worse than the $\Theta(nm)$ algorithm in [EM94], we describe how to solve the decision problem for $\delta_D(\pi, \sigma)$ below, as our algorithm will use this framework, and as the algorithm from [EM94] runs in $\Theta(nm)$ time for any input.

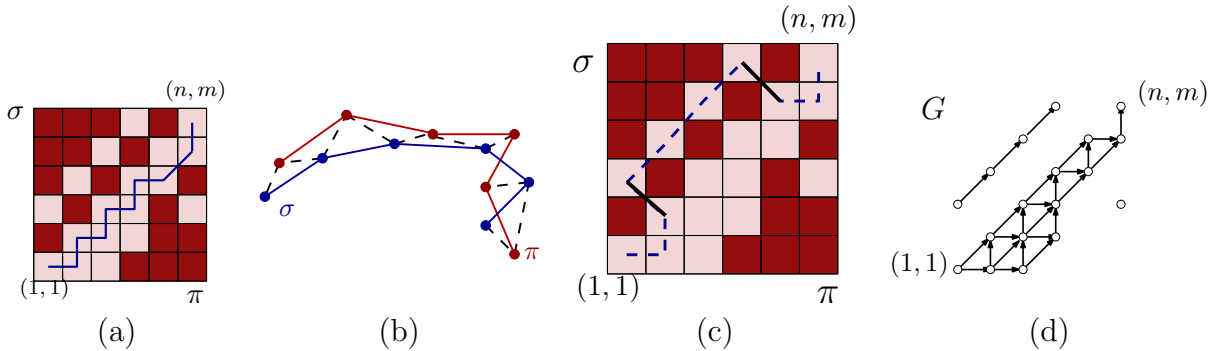


Figure 2: The valid path (solid curve) in the Free-space diagram $D(\pi, \sigma, \delta)$ in (a) corresponds to the order-preserving, complete correspondence (dashed lines) in (b). The path in (c) is not valid, as the two solid segments violate the bi-monotonicity condition. (d) The directed graph corresponding to the white cells in free-space diagram in (a).

Given two polygonal chains π and σ and a distance threshold $\delta \geq 0$, we construct the following *free-space diagram* $D = D(\pi, \sigma, \delta)$; D is an $n \times m$ matrix (grid) and a grid cell $D[i, j]$ has value 1 if

$d(p_i, q_j) \leq \delta$, and value 0 otherwise. We refer to 1-cells as *white* and 0-cells as *black*. The white cells in the i th column (resp., j th row) correspond to the set of vertices of σ (resp., π) whose distance to p_i (resp., q_j) is less than δ . A *viable* path in D is a path connecting $s := D[1, 1]$ to $t := D[n, m]$, visiting only white cells of D , and moving in one step from (i, j) to either $(i, j + 1)$, $(i + 1, j)$, or $(i + 1, j + 1)$. It is easy to check that a complete order-preserving correspondence M induces a viable path in D and vice versa (see Figure 2). Hence the problem of deciding “ $\delta_D(\pi, \sigma) \leq \delta$?” is equivalent to deciding the existence of a viable path in D .

Given D , one can extract a viable path, if it exists, in $\Theta(nm)$ time by a dynamic programming algorithm. Alternatively, one can traverse a directed graph G defined as follows: The nodes of G are the white cells of D . A white cell is connected to its top, right, or top-right neighbor cells by a directed edge, if they are white. See Figure 2 (d). The size of G is the bounded by $|W|$, the number of white cells of D , the in-degree (out-degree) of each node is at most three, and $\delta_D(\pi, \sigma) \leq \delta$ if and only if there is a directed path in G from $(1, 1)$ to (n, m) . That is, testing this condition corresponds to a connectivity check in a directed graph in time $O(|W|)$, once the graph G is given.

Approximations. We say that τ is an ε -approximation of $\delta(\pi, \sigma)$ if

$$(1 - \varepsilon)\delta(\pi, \sigma) \leq \tau \leq (1 + \varepsilon)\delta(\pi, \sigma).$$

We say that an algorithm ε -approximates the decision problem “Is $\delta(\pi, \sigma) \leq \delta$?”, if it returns ‘yes’ whenever $\delta(\pi, \sigma) \leq (1 - \varepsilon)\delta$ and ‘no’ whenever $\delta(\pi, \sigma) \geq (1 + \varepsilon)\delta$. If δ is a $(1 + \varepsilon)$ -approximation of $\delta(\pi, \sigma)$, the algorithm is allowed to return either ‘yes’ or ‘no.’ Such an algorithm is also called an ε -fuzzy decision procedure for $\delta(\pi, \sigma)$.

3 Approximation Algorithms Based on Simplification

In this section, we first introduce a general framework for approximating the discrete Fréchet distance by solving the decision problem approximately. We then present efficient approximation algorithms for two families of common curves based on this framework: the κ -bounded curves and the backbone curves, using curve simplifications, packing arguments, and other observations.

3.1 Approximation via approximate decision problem

Given a set P of N points in \mathbb{R}^d , compute a *well-separated pairs decomposition* (WSPD) of P for a separation parameter 10, which is a collection $\{(A_i, B_i)\}$ of pairs of subsets of P , with the property that (1) for every pair of points $x, y \in P$, there is an index i , so that $x \in A_i$ and $y \in B_i$ and (2) the minimum distance between A_i and B_i is at least 10 times the diameter of either set. One can compute such a collection of size $O(N)$ in $O(N \log N)$ time [CK95]. For every pair (A_i, B_i) in the WSPD, we choose an arbitrary pair of points $p_i \in A_i$ and $q_i \in B_i$ as its representatives. It is easy to check that the distance between any two points $x, y \in P$ is $1/5$ -approximated by the distance between the representatives of the corresponding WSPD pair.

If we want to approximately solve an optimization problem using a decision procedure, where the optimal solution δ^* is one of the distances induced by a pair of points of P , then we can use the above WSPD to extract $O(N)$ values: for each WSPD pair, we take the distance between its representative points. Next, we replace each value x by two values $\frac{4}{5}x$ and $\frac{6}{5}x$, sort the resulting values, and perform a binary search (using the decision procedure) to identify which interval delimited by consecutive values contains δ^* . Let $J = [x, y]$ be the resulting interval; obviously $y \leq \frac{6}{5}x$. We now perform another binary

search on this interval to identify the interval $[x', y']$ containing δ^* with $y' \leq (1 + \varepsilon)x'$, giving rise to an ε -approximation of δ^* . The second binary search invokes the decision procedure $O(\log(1/\varepsilon))$ times.

Interestingly, the decision procedure does not have to be exact, and it can return a *fuzzy* answer, in the sense of last section. An equivalent view of an ε -fuzzy decision procedure is: for a parameter δ , if it returns “no”, then $\delta^* < (1 + \varepsilon)\delta$; otherwise if it returns “yes”, then $\delta^* > (1 - \varepsilon)\delta$. It can be shown that the above binary search can be adapted to work with a fuzzy decision procedure with the same performance guarantees (details omitted and can be found in [Appendix A](#). We summarize:

Theorem 3.1. *Let P be a set of N points in \mathbb{R}^d , and let X be an optimization problem, for which the optimal answer is a distance induced by a pair of points of P . Given an ε -fuzzy decision procedure for X , one can ε -approximate the optimal solution in*

$$O(N \log N + T_{\text{FDECISION}}(N, 1/10) \log N + T_{\text{FDECISION}}(N, \varepsilon/4) \log(1/\varepsilon))$$

time, where $T_{\text{FDECISION}}(N, \epsilon)$ is running time of the fuzzy decision procedure when the required accuracy is ϵ .

Proof: The algorithm is described above. The fuzzy decision procedure can be used with constant accuracy in the stage of the algorithm. Higher accuracy of $\varepsilon/4$ is required only at the second stage, when we perform the binary search over the interval J . ■

On the other hand, observe that there must exist some $p^* \in \pi$ and $q^* \in \sigma$ such that $d(p^*, q^*) = \delta_D(\pi, \sigma)$. In other words, the solution $\delta^* = \delta_D(\pi, \sigma)$ will be one of the distances induced by a pair of points from $P = \{\text{vertices from } \pi \text{ and } \sigma\}$. Hence the above theorem implies that we now only need a fuzzy decision procedure for $\delta_D(\pi, \sigma)$ in order to approximate $\delta_D(\pi, \sigma)$.

3.2 Approximation with simplifications

The remaining question is how to implement fuzzy decision procedure efficiently. One useful heuristic is curve simplification. Below we first describe the particular simplification we use and how it helps in approximating $\delta_D(\pi, \sigma)$. We then show that together with a packing argument and other observations, guaranteed efficiency can be achieved for the two classes of common curves that we investigate.

Greedy simplification. Given a polygonal chain $\pi = \langle p_1, \dots, p_n \rangle$, we simplify π to obtain $\tilde{\pi} = \langle \hat{p}_1, \dots, \hat{p}_k \rangle$, where vertices of $\tilde{\pi}$ form a subsequence of π , with $\hat{p}_1 = p_1$ and $\hat{p}_k = p_n$. More precisely, let $I_\pi(i) = j$ if $\hat{p}_i = p_j \in \pi$; the subscript π is omitted when it is clear from context. We say that $\tilde{\pi}$ μ -simplifies π if (i) $I(i) < I(k)$ for $i < k$ (i.e., order-preserving), and (ii) $d(\hat{p}_i, p_k) \leq \mu$ for any $k \in [I(i), I(i+1))$ (see [Figure 1](#) (c)). (This definition of μ -simplification is slightly different from the standard definition found in the literature.)

We construct a μ -simplification of π , $\tilde{\pi}$, in a greedy manner: Start with $\hat{p}_1 = p_1$. At some stage, suppose we have already computed $\hat{p}_i = p_j$. In order to find $I(i+1)$, we check each vertex of π starting from p_j in order, and stop when we reach the first edge $p_k p_{k+1}$ of π such that $d(p_j, p_k) \leq \mu$ and $d(p_j, p_{k+1}) > \mu$. We set $\hat{p}_{i+1} = p_{k+1}$ and proceed until we reach p_n , at which point we add p_n as the last vertex of $\tilde{\pi}$. The entire procedure takes linear time. By construction, the following observation is straightforward.

Observation 3.2. *For any edge $\hat{p}_i \hat{p}_{i+1}$ in $\tilde{\pi}$, other than the last edge, we have $d(\hat{p}_i, \hat{p}_{i+1}) \geq \mu$.*

Now if we μ -simplify both input curves π and σ to obtain $\tilde{\pi}$ and $\tilde{\sigma}$, we have the following lemma:

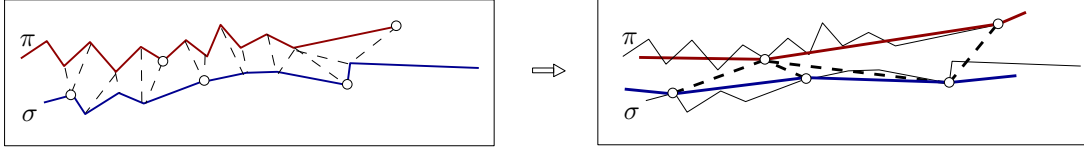


Figure 3: Small empty circles mark vertices of $\tilde{\pi}$ and $\tilde{\sigma}$. Left picture shows part of π and σ and the correspondence M^* (indicated by dashed segments). Right picture shows $\tilde{\pi}$ and $\tilde{\sigma}$ (thick curves) and the induced correspondence for them (thick dashed segments).

Lemma 3.3. $\delta_D(\pi, \sigma) - 2\mu \leq \delta_D(\tilde{\pi}, \tilde{\sigma}) \leq \delta_D(\pi, \sigma) + \mu$.

Proof: We first consider the right-hand inequality. Let $\delta^* = \delta_D(\pi, \sigma)$, and M^* be the complete order-preserving correspondence that produces $\delta_D(\pi, \sigma)$. Obviously, for any pair (p_i, q_j) in M^* , we have that $d(p_i, q_j) \leq \delta^*$. M^* can be modified into a correspondence M between $\tilde{\pi}$ and $\tilde{\sigma}$ as follows: we add the match (\hat{p}_i, \hat{q}_j) to M if and only if (C1) there exists a match $(p_{I_\pi(i)}, q_b) \in M^*$ such that $b \in [I_\sigma(j), I_\sigma(j+1))$, or (C2) there exist a match $(p_a, q_{I_\sigma(j)}) \in M^*$ such that $a \in [I_\pi(i), I_\pi(i+1))$, where I_π (resp. I_σ) maps the indices between π and $\tilde{\pi}$ (resp. σ and $\tilde{\sigma}$) (see Figure 3 for an example). It is easy to verify that M is both complete and order-preserving. By the triangle inequality, we have that $d(\hat{p}_i, \hat{q}_j) \leq d(\hat{p}_i, q_b) + d(q_b, \hat{q}_j)$ for case (C1), implying that $d(\hat{p}_i, \hat{q}_j) \leq d(\hat{p}_i, q_b) + \mu \leq \delta^* + \mu$ (the case for (C2) is symmetric). Since $\delta_D(\tilde{\pi}, \tilde{\sigma})$ should be smaller than the distance induced by M , the right-hand inequality then follows.

The proof for the left-hand inequality is similar but slightly more involved. Details omitted and can be found in the Appendix B. ■

The above lemma implies that if the answer to $\delta_D(\tilde{\pi}, \tilde{\sigma}) \leq \delta$ is ‘yes’, then, $\delta_D(\pi, \sigma) \leq \delta + 2\mu$. If it is ‘no’, then $\delta_D(\pi, \sigma) \geq \delta - \mu$. Thus the decision problem for $\delta_D(\tilde{\pi}, \tilde{\sigma})$ ($2\mu/\delta$)-approximates that of $\delta_D(\pi, \sigma)$. We next show that $\delta_D(\tilde{\pi}, \tilde{\sigma})$ can be answered asymptotically much faster for two special classes of curves, giving rise to efficient fuzzy decision procedure for them.

3.3 Fréchet Distance for κ -bounded curves

Given a polygonal curve π , let $\pi(x, y) \subseteq \pi$ denote the subcurve of π that connects $x \in \pi$ and $y \in \pi$, and $l_\pi(x, y)$ the length of $\pi(x, y)$ along π ; π may be omitted from the subscript when clear. We say that π is κ -straight if $l(x, y) \leq \kappa \cdot d(x, y)$, for any $x, y \in \pi$. Examples of κ -straight curves include *curves with increasing chords* of [Rot94] and *self-approaching curves* of [AAI⁺01]. As defined by Alt et al. [AKW04], π is κ -bounded if $\pi(x, y) \subseteq B(x, \frac{\kappa}{2}d(x, y)) \cup B(y, \frac{\kappa}{2}d(x, y))$, for all $x, y \in \pi$, where $B(x, r)$ is the radius- r Euclidean ball centered at x and where we have slightly abused the notation by treating a curve section as a point set. See Figure 1 (b) for an illustration in two dimensions. Every κ -straight curve is κ -bounded.

We now describe how to construct an ε -fuzzy decision procedure for the problem “ $\delta_D(\pi, \sigma) \leq \delta$?”, where one curve, say σ , is κ -bounded. We first μ -simplify π and σ into $\tilde{\pi}$ and $\tilde{\sigma}$ respectively, using $\mu := \varepsilon\delta/2$. By Lemma 3.3, the decision problem for $\delta_D(\tilde{\pi}, \tilde{\sigma})$ is an ε -fuzzy decision procedure for $\delta_D(\pi, \sigma)$. Hence we now focus on checking whether $\delta_D(\tilde{\pi}, \tilde{\sigma}) \leq \delta$. Let n, m, r, s be the size of $\pi, \sigma, \tilde{\pi}$, and $\tilde{\sigma}$ respectively; $r = O(n)$ and $s = O(m)$.

Decision problem for $\delta_D(\tilde{\pi}, \tilde{\sigma})$. Let \mathcal{D} be the free-space diagram for $\tilde{\pi}$ and $\tilde{\sigma}$ with respect to δ . Recall that $\delta_D(\tilde{\pi}, \tilde{\sigma}) \leq \delta$ if there exists a viable path in \mathcal{D} which can be computed in $O(|W|)$ time once W , the set of white cells of \mathcal{D} are given. We first bound the size of W .

For every $\hat{p} \in \tilde{\pi}$, let $N(\hat{p})$ be the set of points from $\tilde{\sigma}$ contained in $B(\hat{p}, \delta)$. Obviously, $|W| = \sum_{\hat{p} \in \tilde{\pi}} |N(\hat{p})|$. Consider any two points $q_1, q_2 \in \tilde{\sigma}$ that lie in $B(\hat{p}, \delta)$ for some $\hat{p} \in \tilde{\pi}$. There are two cases: (i) $q_1 q_2$ is an edge of $\tilde{\sigma}$ and (ii) otherwise. For case (i), we have that $d(q_1, q_2) \geq \mu$ by [Observation 3.2](#). For case (ii), we know that

$$\sigma(q_1, q_2) \subseteq B(q_1, \frac{\kappa}{2}d(q_1, q_2)) \cup B(q_2, \frac{\kappa}{2}d(q_1, q_2)),$$

as σ is κ -bounded. Furthermore, let $q_1 q \subset \tilde{\sigma}$ be the edge with $q \in \sigma(q_1, q_2)$; $d(q_1, q) \geq \mu$ by [Observation 3.2](#). It then follows that $(\kappa/2)d(q_1, q_2) \geq \mu$ and therefore $d(q_1, q_2) \geq 2\mu/\kappa$. Hence $N(\hat{p}) = O((\kappa\delta/\mu)^d)$ by a straightforward packing argument. This means that the number of white cells is $|W| = O(s(\kappa\delta/\mu)^d) = O(n(\kappa\delta/\mu)^d)$ given that σ is a κ -bounded curve.

We still need to compute $N(\hat{p})$ efficiently, that is, to enumerate the set of vertices of $\tilde{\sigma}$ contained in $B(\hat{p}, \delta)$ for every $\hat{p} \in \tilde{\pi}$. This can be done by a spherical range query. As there are no known efficient algorithms for spherical range queries, we instead first perform a β -approximate range query of $B(\hat{p}, \delta)$ among all vertices from $\tilde{\sigma}$, such that vertices lying completely inside $B(\hat{p}, \delta)$ are guaranteed to be retrieved, those completely outside $B(\hat{p}, (1+\beta)\delta)$ will not be reported, while those in-between may or may not be returned. By the same packing argument as above, it is easy to verify that the number of vertices returned is still bounded by $O((\frac{(1+\beta)\kappa\delta}{\mu})^d)$. We then inspect each vertex returned, and only mark the corresponding cell in \mathcal{D} white when it indeed lies in $B(\hat{p}, \delta)$.

We preprocess $\tilde{\sigma}$ into a data structure of size $O(s) = O(m)$, using $O(s)$ preprocessing time, such that the resulting data-structure answers β -approximate range query for $B(\hat{p}, \delta)$ in $O(1/\beta^d)$ time. This can be easily achieved by constructing a grid of appropriate size (which is $\beta\delta$), throwing the points of $\tilde{\sigma}$ into this grid (using hashing). Next, an approximate spherical range query is no more than probing all the grid cells that intersects the query ball. The number of cells being probed in a single query is $O(1/\beta^d)$. Therefore the set of white cells in \mathcal{D} can be computed in $O(r + s + r(\kappa\delta/\mu)^d)$ time by choosing $\beta > 0$ to be a small constant, say $\beta = 1/2$.

Putting everything together, we have an ε -fuzzy decision procedure for $\delta_D(\pi, \sigma)$ that runs in $O(n + m + n\kappa^d/\varepsilon^d)$ time and space in \mathbb{R}^d . By [Theorem 3.1](#), we have that:

Lemma 3.4. *An ε -approximation of $\delta_D(\pi, \sigma)$ for a polygonal curve π and a κ -bounded curve σ , of size n and m respectively, can be computed in $O((m + n\kappa^d/\varepsilon^d) \log(n/\varepsilon))$ time and $O(n + m + n\kappa^d/\varepsilon^d)$ space in d dimensions.*

3.4 Fréchet Distance for Protein Backbones

In molecular biology, it is common to model a protein backbone by a polygonal chain, where each C_α atom becomes a vertex, and each edge represents a covalent bond between two consequent amino acids. All the bonds have approximately the same bond length, and no two atoms (thus vertices) can get too close due to van der Waals interactions. This is the motivation behind the study of the *backbone* curves, which have the following properties:

- P1. For any two non-consecutive vertices u and v of the curve, $d(u, v) \geq 1$,
- P2. Every edge of the curve has length l such that $c_1 \leq l \leq c_2$, where $c_1, c_2 > 0$ are constants.

We remark that although proteins lie in three dimensional space, there are simplified models for protein backbones in both two and three dimensions, such as the lattice model which has been widely studied to understand the mechanism behind protein folding [[GIP99](#), [KS94](#)].

Now suppose we are given backbone curves π and σ in \mathbb{R}^d . Given a distance threshold $\delta \geq 0$, we want to know whether $\delta_D(\pi, \sigma) \leq \delta$. We μ -simplify π and σ to obtain $\tilde{\pi}$ and $\tilde{\sigma}$ as in the previous case, for $\mu = \varepsilon\delta/2$, and construct the free-space diagram \mathcal{D} for $\tilde{\pi}$ and $\tilde{\sigma}$ with respect to δ . \mathcal{D} is an $r \times s$ grid, where by [Observation 3.2](#) and property P2, $r = |\tilde{\pi}| \leq c_2 n/\mu$ and $s = |\tilde{\sigma}| \leq c_2 m/\mu$. Once \mathcal{D} is given, the decision problem can be solved in time proportional to $|W|$, where W is the set of white cells in \mathcal{D} .

The set of white cells W . A straightforward bound for $|W|$ is $O(\min\{r\delta^d, s\delta^d\})^1$, as by the packing argument and property P1, there are at most $O(\delta^d)$ vertices lying in δ -neighborhood of any vertex of $\tilde{\pi}$ and $\tilde{\sigma}$. If $\delta < 1$, then the number of white cells is $O(n + m)$. Hence we now assume that $\delta \geq 1$.

We can improve this bound by a more careful counting analysis. Assume without loss of generality that $r \leq s$. For any vertex $\hat{p} \in \tilde{\pi}$ and its δ -neighborhood $B(\hat{p}, \delta)$, let $E(\hat{p})$ be the set of edges of $\tilde{\sigma}$ intersecting the ball $B(\hat{p}, \delta)$. The number of vertices of $\tilde{\sigma}$ in $B(\hat{p}, \delta)$ can be upper bounded by $O(|E(\hat{p})|)$. Furthermore, given any edge $e = (\hat{q}_i, \hat{q}_{i+1}) \in \tilde{\sigma}$, let $\sigma(e) = \sigma(q_{I_q(i)}, q_{I_q(i+1)})$ (that is, subchain $\sigma(e) \subseteq \sigma$ is simplified into edge e in chain $\tilde{\sigma}$). $E(\hat{p})$ can be partitioned into two sets: (i) $E_1 = \{e \in E(\hat{p}) \mid \sigma(e) \subseteq B(\hat{p}, \delta)\}$, and (ii) $E_2 = \{e \in E(\hat{p}) \mid \text{at least a vertex of } \sigma(e) \text{ lies outside } B(\hat{p}, \delta)\}$.

By property P2, we know that the number of vertices in $\sigma(e)$ is at least μ/c_2 for any $e \in \tilde{\sigma}$. Therefore $|E_1| = O(c_2\delta^d/\mu)$. On the other hand, for every edge $e \in E_2$, there is at least one vertex of $\sigma(e)$ that lies in the spherical shell of $B(\hat{p}, \delta + c_2) \setminus B(\hat{p}, \delta)$, as the length of edges in σ is at most c_2 . Since the volume of this spherical shell is $O(c_2(c_2 + \delta)^{d-1})$, the size of E_2 is bounded by $O((c_2(c_2 + \delta)^{d-1}/(c_1^{d-1})))$. Therefore, we have that $|E(\hat{p})| = |E_1| + |E_2| = O(\delta^{d-1} + \delta^d/\mu)$. Summing it over all r vertices of $\tilde{\pi}$, we have that $|W| = O(\frac{n}{\mu}(\delta^{d-1} + \delta^d/\mu))$. Furthermore, since this number cannot exceed the size of \mathcal{D} which is $O(rs) = O(nm/\mu^2)$, we have $|W| = \min\{nm/\mu^2, O(\frac{n}{\mu}(\delta^{d-1} + \delta^d/\mu))\}$. Note that $|W|$ is maximized when the two balancing terms are equal: $\frac{nm}{\varepsilon^2\delta^2} = \frac{\delta^{d-2}}{\varepsilon^2}$, that is, when $\delta = m^{1/d}$. This implies that $|W| = O(nm^{1-2/d}/\varepsilon^2)$.

We still need to compute these white cells of \mathcal{D} efficiently. Similar to the case for κ -bounded curves, we preprocess $\tilde{\sigma}$ into a data structure of size $O(s)$, using $O(s)$ preprocessing time, such that the resulting data structure answers β -approximate range query for $B(\hat{p}, \delta)$ in $O(1/\beta^d)$ time, for a small constant $\beta > 0$, say $\beta = 1/2$. We then check all vertices returned by this approximate range query, and keep only those indeed contained in $B(\hat{p}, \delta)$. Overall, we can compute all white cells in $O(r + s + |W|)$ time and space, thus can answer the decision problem “Is $\delta_D(\tilde{\pi}, \tilde{\sigma}) \leq \delta$?” in the same time and space. Putting everything together, we have:

Lemma 3.5. *Given two backbone curves of sizes n and m , respectively, we can develop an ε -fuzzy decision procedure for $\delta_D(\pi, \sigma)$ w.r.t. δ that runs in $O((n + m) + \frac{1}{\varepsilon^2}nm^{1-2/d})$ time and space. In particular, the time complexity is $O(n + m/\varepsilon^2)$ when $d = 2$, and $O(n + m + nm^{1/3}/\varepsilon^2)$ when $d = 3$.*

Finally, for backbone curves, in order to approximate $\delta_D(\pi, \sigma)$, one can use a binary search procedure (described in [Appendix C](#) instead of the approach using WSPD as described earlier. The advantage of the binary search procedure is that all results can then be extended for the continuous case $\delta_F(\pi, \sigma)$ by more careful and involved packing arguments. We conclude with the following theorem.

Theorem 3.6. *Given two backbone curves π and σ of n and m vertices respectively, we can compute an ε -approximation of $\delta_F(\pi, \sigma)$ in $O(\frac{(n+m)}{\varepsilon^3} \log(nm))$ time in two dimensions, and $O(\frac{1}{\varepsilon^3}nm^{1/3} \log(nm))$ time in three dimensions.*

¹In the following, the big O notation sometimes hide factors depending on constants c_1 and c_2 .

4 Pseudo–Output-Sensitive Algorithm

In this section, given curves π and σ of size n and m , respectively, we present a pseudo-output-sensitive algorithm for computing $\delta_D(\pi, \sigma)$ for general curves. Although the worst case complexity may still be $\Theta(nm)$, we believe that the observation made within should help to produce efficient (possibly approximate) algorithms for Fréchet distance in practice. In what follows, we provide results for L_∞ norm (which provides a constant factor approximation for optimal solution under L_2 norm). The time complexity for exact computation under L_2 norm is quite messy and omitted.

Suppose we have an algorithm that answer the following *select-distance* query in $B(N)$ time: given a set of N points P and a rank k , what is $\text{RANK}_d(k)$, the k th smallest distance among all pair-wise distances from P . Now given an algorithm to solve the decision problem “Is $\delta^* = \delta_D(\pi, \sigma) \leq \delta$?” in time $A(n+m)$, we can find the optimal solution δ^* in $O((A(n+m) + B(n+m)) \log(nm))$ time by querying $\text{RANK}_d(k)$ among $n+m$ points in a binary search manner². For L_∞ norm, the distance-selection problem can be solved in $O(dN \log^{d-1} N)$ in \mathbb{R}^d [Sal89]. For the decision problem, a straightforward bound for time complexity A is $|W|$ plus the time to compute W , where W is the set of white cells in the free-space diagram $D = D(\pi, \sigma, \delta)$ for a threshold $\delta > 0$. Below we provide a tighter bound for A although its worst-case complexity is still $\Theta(nm)$.

Switching cells. Given an $n \times m$ map D with respect to some threshold δ , a *switching cell* is a white cell whose immediate neighbor above or below it is black. So if $D[i, j]$ is a switching cell, then the edge $q_j q_{j+1} \subset \sigma$ (or $q_j q_{j-1}$) intersects the boundary of $B(p_i, \delta)$ exactly once (one endpoint must lie inside and one must be outside). For a vertex $p \in \pi$, while the set of white cells involving p correspond those vertices from q falling inside $B(p, \delta)$, the switching cells involving p correspond to those vertices inside $B(p, \delta)$ with one incident edge crossing the boundary of $B(p, \delta)$. Let $\S = \S(\pi, \sigma, \delta)$ denote the set of switching cells of $D(\pi, \sigma, \delta)$. Although in worst case $|\S| = \Omega(|W|) = \Omega(nm)$, we expect it to be much smaller than $|W|$ in practice. For example, consider the case when vertices of σ form lines of a cubic lattice of size $n^{1/3} \times n^{1/3} \times n^{1/3}$ and δ is roughly $n^{1/3}/2$. For a vertex p at the center of this cube, the number of white cells in the corresponding column in D is $\Theta(n)$, while the number of switching cells is $\Theta(n^{2/3})$. The remaining questions are (i) how to compute the set of switching cells $\S(\pi, \sigma, \delta)$ and (ii) how to solve the decision problem once \S is given.

Decision problem with \S . Once the set of switching cells is given, we can solve the decision problem in $O(|\S|)$ time and space as follows. Instead of representing D explicitly, we now represent each column of D , $C[i]$ for $1 \leq i \leq n$, as a set of ordered intervals, where each interval corresponds to a maximal set of consecutive white cells in this column. Obviously, the endpoints of these intervals are exactly the switching cells. Let $V[i]$ be the set of ordered intervals, each representing a maximal set of cells in the i th column reachable from $D[1, 1]$, and $|C[i]|$ and $|V[i]|$ the number of intervals in $C[i]$ and $V[i]$, respectively. Easy to see that $|V[i]| \leq |C[i]|$, because all cells covered by intervals from $V[i]$ are white, and because if any cell c from an interval $I \in C[i]$ is in some interval $J \in V[i]$, then all cells of I above c should also be covered by J . Our algorithm scans D from left to right (i.e, from column 1 to column n), and at the i th round, we compute $V[i]$ by merging $C[i]$ and $V[i-1]$ in $O(|V[i-1]| + |C[i]|)$ time using a merge-sort like procedure (see details in [Appendix D](#)).

Computing \S . Given p and δ , let $\S(p, \delta)$ denote the set of edges from σ “crossing” the boundary of $B(p, \delta)$. Here by *crossing*, we mean that one endpoint of the edge is inside $B(p, \delta)$ and one is outside (so

²In some sense, our previous approach using WSPD is performing implicit approximate distance selection.

it is not the usual segment/ball intersection problem). To compute \S , we need to perform n *edge/ball crossing queries*, one for each vertex from π . Under the L_∞ norm, the basic operation is in fact an edge/cube crossing query, where all cubes are congruent. We can preprocess the set of edges by building a range-search tree for their endpoints (similar to the multi-level data structure for orthogonal range reporting problem). The entire data structure has size $O(m \log^{2d} m)$ and given a cube, the set of edges crossing it can be reported in $O(\log^{2d} m + k)$ where k is the number of such edges.

Putting everything together, we conclude with the following theorem:

Theorem 4.1. *Given two arbitrary polygonal curves π and σ in \mathbb{R}^d , with n and m vertices, respectively, one can compute $\delta_D(\pi, \sigma)$ under L_∞ -norm, in $O((\Phi + (n+m) \log^{2d}(nm)) \log(nm))$ time and $O(\Phi + (n+m) \log^{2d}(nm))$ space, where Φ is an upper bound of the number of switching cells for any threshold δ .*

We remark that for L_2 norm, the running time is $\tilde{O}(\Phi + (n+m)^{4/3} \log(nm))$ for $d = 2$ and $\tilde{O}((\Phi + (n+m)^{2-1/2^d}) \log(nm))$ time for $d > 2$. The edge/ball crossing query required by computing \S can be converted into an segment/hyperplane query in one dimension higher. It is less practical as the solution involves heavy machinery. Nevertheless, if approximation is allowed, one can use the idea from [Theorem 3.1](#) as well as multi-dimensional range trees to obtain an ε -approximation algorithm in $O(\Phi + \text{polylog } n)$ time and space where polylog depends on both ε and d .

5 Conclusions and Discussion

In this paper, we considered the problem of computing discrete Fréchet distance between two polygonal curves either approximately or exactly. Our main contribution is a simple approximation framework that leads to efficient ε -approximation algorithms for two families of common curves: the κ -bounded curves and the backbone curves. We also consider the exact algorithm for general curves, and proposed a pseudo-output-sensitive algorithm by observing that only a subset of the white cells from the free-space diagram are necessary for the decision problem. It will be interesting to investigate whether there are families of curves that are guaranteed to have small Φ , which is the upper bound on the number of switching cells.

We feel that for general curves, it might be hard to develop algorithms that are significantly sub-quadratic in worst case, given that no such algorithm exists for a related and widely studied problem, the edit distance for strings. Hence our future directions will focus on practical variants of Fréchet distance so that one can handle outliers and/or partial matching, or so that one can perform efficient multiple-curve alignments. Another important direction is to develop efficient (approximation) algorithm for computing smallest Fréchet distance under rigid motions (in particular rotations).

Postscript. Since the appearance of this paper in ESA 2006 [\[AHK⁺06\]](#) a lot of research was done on related problems. Driemel et al. [\[DHW12\]](#) introduced the notion of c -packed curves, and showed a near linear time algorithm for such curves. Bringmann [\[Bri14\]](#) proved that Fréchet distance can not be computed exactly in subquadratic time under the SETH hypothesis. There is more recent research on the Fréchet distance, but surveying it is outside the scope of this note.

References

- [AAI⁺01] O. Aichholzer, F. Aurenhammer, C. Icking, R. Klein, E. Langetepe, and G. Rote. Generalized self-approaching curves. *Disc. App. Math.*, 109(1-2):3–24, 2001.

- [AB05] H. Alt and M. Buchin. Semi-computability of the Fréchet distance between surfaces. In *Proc. 21st Euro. Workshop on Comput. Geom.*, pages 45–48, 2005.
- [AERW03] H. Alt, A. Efrat, G. Rote, and C. Wenk. Matching planar maps. *J. Algorithms*, 49:262–283, 2003.
- [AG95] H. Alt and M. Godau. Computing the Fréchet distance between two polygonal curves. *Internat. J. Comput. Geom. Appl.*, 5:75–91, 1995.
- [AHK⁺06] B. Aronov, S. Har-Peled, C. Knauer, Y. Wang, and C. Wenk. Fréchet distance for curves, Revisited. In *Proc. 14th Annu. European Sympos. Algorithms (ESA)*, pages 52–63, 2006.
- [AHMW05] P. K. Agarwal, S. Har-Peled, N. Mustafa, and Y. Wang. Near-linear time approximation algorithms for curve simplification in two and three dimensions. *Algorithmica*, 42:203–219, 2005.
- [AKW01] H. Alt, C. Knauer, and C. Wenk. Matching polygonal curves with respect to the fréchet distance. In *Proc. 18th Internat. Sympos. Theoret. Asp. Comp. Sci.*, pages 63–74, 2001.
- [AKW04] H. Alt, C. Knauer, and C. Wenk. Comparison of distance measures for planar curves. *Algorithmica*, 38(1):45–58, 2004.
- [AST94] P. K. Agarwal, M. Sharir, and S. Toledo. Applications of parametric searching in geometric optimization. *J. Algorithms*, 17:292–318, 1994.
- [BBW06] K. Buchin, M. Buchin, and C. Wenk. Computing the Fréchet distance between simple polygons in polynomial time. In *Proc. 22nd Annu. Sympos. Comput. Geom. (SoCG)*, pages 80–87, 2006.
- [Bri14] K. Bringmann. Why walking the dog takes time: Frechet distance has no strongly sub-quadratic algorithms unless SETH fails. In *Proc. 55th Annu. IEEE Sympos. Found. Comput. Sci. (FOCS)*, pages 661–670, 2014.
- [CK95] P. B. Callahan and S. R. Kosaraju. A decomposition of multidimensional point sets with applications to k -nearest-neighbors and n -body potential fields. *J. Assoc. Comput. Mach.*, 42:67–90, 1995.
- [CM05] M. Clausen and A. Mosig. Approximately matching polygonal curves with respect to the Fréchet distance. *Comput. Geom. Theory Appl.*, 30:113–127, 2005.
- [DHW12] A. Driemel, S. Har-Peled, and C. Wenk. Approximating the Fréchet distance for realistic curves in near linear time. *Discrete Comput. Geom.*, 48:94–127, 2012.
- [EGH⁺02] A. Efrat, L. J. Guibas, S. Har-Peled, J. S.B. Mitchell, and T.M. Murali. New similarity measures between polylines with applications to morphing and polygon sweeping. *Discrete Comput. Geom.*, 28:535–569, 2002.
- [EGHM01] A. Efrat, L. J. Guibas, S. Har-Peled, and T. M. Murali. Morphing between polylines. In *Proc. 12th ACM-SIAM Sympos. Discrete Algs. (SODA)*, pages 680–689, 2001.
- [EM94] T. Eiter and H. Mannila. Computing discrete Fréchet distance. Tech. Report CD-TR 94/64, Christian Doppler Lab. Expert Sys., TU Vienna, Austria, 1994.

- [GIP99] D. Goldman, S. Istrail, and C. H. Papadimitriou. Algorithmic aspects of protein structure similarity. In *Proc. 40th Annu. IEEE Sympos. Found. Comput. Sci. (FOCS)*, pages 512–522, 1999.
- [Ind02] P. Indyk. Approximate nearest neighbor algorithms for Fréchet distance via product metrics. In *Proc. 18th Annu. Sympos. Comput. Geom. (SoCG)*, pages 102–106, 2002.
- [KHM⁺98] S. Kwong, Q. H. He, K. F. Man, K. S. Tang, and C. W. Chau. Parallel genetic-based hybrid pattern matching algorithm for isolated word recognition. *Int. J. Pattern Recog. Art. Intel.*, 12(5):573–594, August 1998.
- [KKS05] M.S. Kim, S.W. Kim, and M. Shin. Optimization of subsequence matching under time warping in time-series databases. In *Proc. ACM symp. Applied comput.*, pages 581–586, 2005.
- [KP99] E. J. Keogh and M. J. Pazzani. Scaling up dynamic time warping to massive dataset. In *Proc. of the Third Euro. Conf. Princip. Data Mining and Know. Disc.*, pages 1–11, 1999.
- [KS94] A. Kolinski and J. Skolnick. Monte carlo simulations of protein folding: Lattice model and interaction scheme. In *Proteins*, volume 18, pages 338–352, 1994.
- [PP90] M. Parizeau and R. Plamondon. A comparative analysis of regional correlation, dynamic time warping, and skeletal tree matching for signature verification. *IEEE Trans. Pattern Anal. Mach. Intell.*, 12(7):710–717, 1990.
- [Rot94] G. Rote. Curves with increasing chords. *Math. Proc. Camb. Phil. Soc.*, 115:1–12, 1994.
- [Rot05] G. Rote. Computing the Fréchet distance between piecewise smooth curves. Technical Report ECG-TR-241108-01, Freie Universitat, Berlin, May 2005. To appear in *Comput. Geom. Theory Appl.*
- [Sal89] J. S. Salowe. L_∞ interdistance selection by parametric search. *Inform. Process. Lett.*, 30:9–14, 1989.
- [Wen02] C. Wenk. *Shape Matching in Higher Dimensions*. PhD thesis, Dept. of Comput. Sci., Freie Universitat, Berlin, 2002.

A Approximation via fuzzy decision procedure

Given an β -fuzzy decision procedure $\text{ApprDecision}(\delta, \beta)$ for deciding whether $\delta^* \leq \delta$, we combine it with the WSPD approach described in [Section 3.1](#) to compute an ε -approximation of δ^* . In particular, construct the $O(n)$ distances using WSPD as before, and perform a binary search by querying $\text{ApprDecision}(v, 1/10)$ among these distances to identify the interval $\mathcal{J} = [x, y]$ such that $\text{ApprDecision}(x, 1/10)$ returns “no” while $\text{ApprDecision}(y, 1/10)$ returns “yes”. Easy to verify that $a = \frac{4}{5}x < \delta^* < \frac{7}{5}x = b$. We then start with a pair $k_l = a/(1 + \varepsilon)$, and $k_h = b/(1 - \varepsilon)$, and perform a standard binary search while always maintaining that $\text{ApprDecision}(k_l, \varepsilon/4)$ returns “no” and $\text{ApprDecision}(k_h, \varepsilon/4)$ returns “yes” until $k_h - k_l \leq (b - a)\varepsilon/3$. It is easy to verify that the invariant holds when we start, and the number of

iterations is at most $O(\log(1/\varepsilon))$. Furthermore, because of the invariant that we maintain, we have $(1 - \varepsilon/4)k_l \leq \delta^* \leq (1 + \varepsilon/4)k_h$ and $k_h - k_l \leq (b - a)\varepsilon/2 < \delta^*\varepsilon/2$. It then follows that when $\varepsilon < 1$,

$$\delta^* \leq (1 + \varepsilon/4)k_h \leq (1 + \varepsilon/4)(k_l + \delta^*\varepsilon/2) \rightarrow \delta^* \leq (1 + \varepsilon/4)k_l / (1 - (1 + \varepsilon/4)\varepsilon/2) \leq (1 + \varepsilon)k_l.$$

This implies that k_l is an ε -approximation of δ^* . Hence we can use a fuzzy decision procedure to approximate δ^* .

B Left-inequality of Lemma 3.3

Let C^* be the complete order-preserving correspondence that produce $\delta_D(\tilde{\pi}, \tilde{\sigma})$. We now modify it into a correspondence C between π and σ as follows: First we add all matches $(p_{I_\pi(i)}, q_{I_\sigma(j)})$ to C if $(\hat{p}_i, \hat{q}_j) \in \mathcal{C}^*$. Next, we take each pair of consecutive matches. There are three cases as illustrated in Figure 4. The first two are symmetric, and we simply add matches $(p_{I_\pi(i)}, q_k)$ (resp. $(p_k, q_{I_\sigma(j)})$) into C

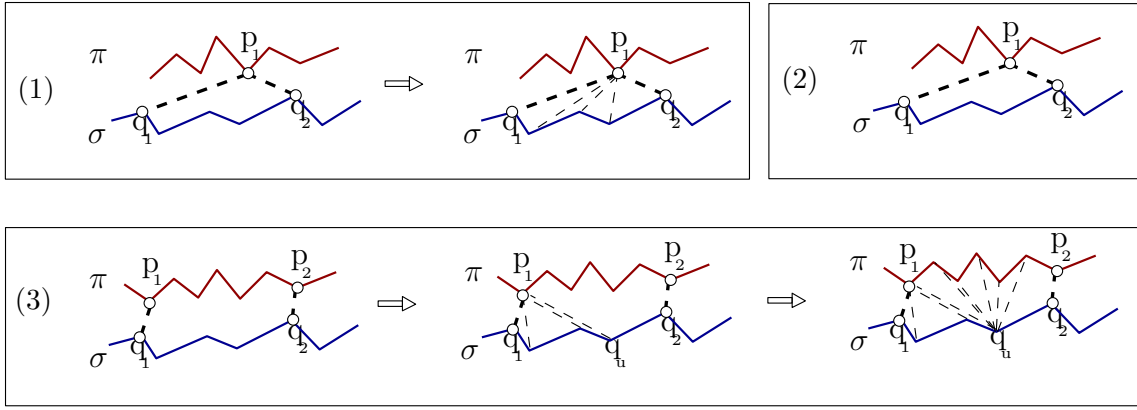


Figure 4: Small empty circles mark vertices of $\tilde{\pi}$ and $\tilde{\sigma}$. Three cases for two consecutive pairs from C^* between $\tilde{\pi}$ and $\tilde{\sigma}$. For case (3), we first add all correspondences between p_1 and all vertices between q_1 and q_2 along σ , we then add all correspondences between q_u and vertices between p_1 and p_2 .

for $k \in (I_\sigma(j), I_\sigma(j+1))$ (resp. $k \in (I_\pi(i), I_\pi(i+1))$). For the third case, we add all matches of the form $(p_{I_\pi(i)}, q_k)$ for $k \in (I_\sigma(j), I_\sigma(j+1))$, and of the form (p_k, q_u) for $k \in (I_\pi(i), I_\pi(i+1))$ and $u = I_\sigma(j+1) - 1$. It is easy to verify that the resulting matching M is both complete and order-preserving. Furthermore, by triangle inequality, each match (p_i, q_j) added for the first two cases satisfies $d(p_i, q_j) \leq \delta_D(\tilde{\pi}, \tilde{\sigma}) + \mu$; while an edge (p_k, q_u) added in last case satisfies

$$d(p_k, q_u) \leq d(p_k, p_{I_\pi(i)}) + d(p_{I_\pi(i)}, q_{I_\sigma(j)}) + d(q_{I_\sigma(j)}, q_u) \leq \delta_D(\tilde{\pi}, \tilde{\sigma}) + 2\mu.$$

This proves the left-hand inequality in Lemma 3.3.

C Approximating $\delta_D(\pi, \sigma)$ for backbone curves

Let $\text{ApprDecision}(\pi, \sigma, \delta, \varepsilon)$ denote the ε -fuzzy decision procedure for $\delta_D(\pi, \sigma)$ for two backbone curves π and σ . In order to find an ε -approximation of $\delta_D(\pi, \sigma)$, we can simply use Theorem 3.1. However, for this particular case, we can have a much simpler binary search procedure within similar time/space complexity that avoids the construction of WSPD.

In particular, if $\delta_D(\pi, \sigma) < \beta$, for some constant β , say $\beta = 1$, then we know that there exist a pair of vertices $p^* \in \pi$ and $q^* \in \sigma$ such that $d(p^*, q^*) = \delta_D(\pi, \sigma) < \beta$. We collect the set T of all pairs between π and σ with distance smaller than β . By similar packing argument as in [Section 3.4](#) (in computing white cells), $|T| = O(n + m)$ and we can compute T in the same time/space. We then simply perform a binary search among T to locate (p^*, q^*) and compute $\delta_D(P, Q)$ exactly in $O((n + m) \log(nm))$ time. We call this procedure `ExactFSmall`(P, Q, ε, β).

```

ALGORITHM  ApprFBackbone( $P, Q, \varepsilon$ )
begin
  Set  $\delta_o = \delta_n = 1$ , yes = 0, no = 0.
  while ( yes == 0 or no == 0 ) do
    if  $\delta < 1$  then
      return ExactFSmall( $P, Q, \varepsilon, 2$ )
    end if
    if ApprDecision( $\delta_n, \varepsilon/3$ ) == ‘yes’ then
      set yes = 1,  $\delta_o = \delta_n$ ,  $\delta_n = \delta_o/(1 + \varepsilon/3)$ 
    else
      set no = 1,  $\delta_o = \delta_n$ ,  $\delta_n = (1 + \varepsilon/3)\delta_o$ 
    end if
  end while
  return  $\delta_o$ 
end

```

Figure 5: Algorithm `ApprFBackbone`(P, Q, ε) computes an ε -approximation of $\delta_D(P, Q)$ for two backbone curves. Subroutine `ExactFSmall`(P, Q, ε, β) computes $\delta_D(P, Q)$ exactly if $\delta_D(P, Q) < \beta$.

For the case when $\delta_D(\pi, \sigma) \geq \beta$, we perform a different search procedure as described in [Figure 5](#). Easy to verify that **while** loop can be called at most $O(\log_{1+\varepsilon}(n + m)) = O((\log(n + m))/\varepsilon)$ time, as obviously $\delta_D(\pi, \sigma) \leq c_2(n + m)$ for backbone curves. To see that the output of the algorithm is indeed an ε -approximation of $\delta_D(\pi, \sigma)$, observe that when the algorithm terminates, the sequence of answers from `ApprDecision`() is either a sequence of “yes” followed by **one** “no”, or a sequence of “no” followed by **one** “yes”. Let assume that we have the first case (the second is symmetric). Suppose the output of the algorithm is $\bar{\delta}$, then we have that $(1 - \varepsilon/3)\bar{\delta} \leq \delta^* = \delta_D(\pi, \sigma)$. In the previous iteration of the **while** loop, $\delta_n = \bar{\delta}(1 + \varepsilon/3)$, as the answer then was “yes”. Hence we have that $\delta^* \leq (1 + \varepsilon/3)\delta_n = (1 + \varepsilon/3)^2\bar{\delta} \leq (1 + \varepsilon)\bar{\delta}$ if $\varepsilon < 1$. This implies that $\bar{\delta}$ ε -approximates δ^* . Hence [Theorem 3.6](#) follows.

D Decision problem with switching cells §

The only step unexplained is how to compute $V[i]$ by merging $C[i]$ and $V[i - 1]$ in $O(|V[i - 1]| + |C[i]|)$ time. This can be achieved by a bottom-up scanning for $V[i - 1]$ and $C[i]$ simultaneously. More specifically, given $V[i - 1]$ and $C[i]$, we can sort the endpoints of their intervals in $O(|V[i - 1]| + |C[i]|)$ time using merge sort. We process them in order and maintain the partial $V[i]$ at any time. For sake of simplicity, we call an endpoint a *L-point* (resp. *H-point*) of $V[i - 1]$ or $C[i]$ if it is the low-endpoint (resp. higher endpoint) of some interval from $V[i - 1]$ or $C[i]$. The pseudo-code is shown in [Figure 6](#), where $V = V[i - 1]$, $C = C[i]$, and the output is $X = V[i]$. It is easy to verify the correctness of the algorithm, and the running time is proportional to the sum of interval lists being merged.

```

ALGORITHM  mergeColumn( $V, C$ )
begin
    Set potentialReachFlag = 0 and potentialStartFlag = 0
    Sort  $H$ , the set of endpoints from  $V$  and  $C$ 
    for (  $i = 1; i < |H|; i++$  ) do
        if (  $H[i]$  is L-point of  $V$  ) then
            Set potentialReachFlag = 1
            if ( potentialStartFlag == 1 )
                then    Add  $H[i]$  as L-point for  $X$ 
            else if (  $H[i]$  is H-point of  $V$  ) then
                Set potentialReachFlag = 0
            else if (  $H[i]$  is L-point of  $C$  ) then
                Set potentialStartFlag = 1
                if ( potentialReachFlag = 1 )
                    then    Add  $H[i]$  as L-point for  $X$ 
            else if (  $H[i]$  is H-point of  $C$  ) then
                Set potentialStartFlag = 0
                Add  $H[i]$  as H-point for  $X$ 
        end for
    end

```

Figure 6: Algorithm to compute $V[i]$ (i.e, X) from $V[i - 1]$ (i.e, V) and $C[i]$ (i.e, C).